their encoded location PLs are fed into the matching service. This is done in the following steps - first the LBS notifies the mobile operator of the physical locations for each encoded business PI it caters to (indicated as Step 1 in Figure 4.3), followed by the mobile operator notifying the matching service of each PI with the corresponding PL values for each of the locations of that business (Step 2). The mobile user also relies on an initial registration phase with the LBS to get the PIs corresponding to the businesses and users of interest (Step 3). Before sending a LBS query, the mobile user contacts the mobile operator for the PL corresponding to its current location (Step 4). It can then initiate a LBS query, the generic form of which is a 3-tuple <user's-PI, user's-PL, desired-PI-list> (Step 5). Note that the 'user's PI' field is required only when the mobile user wants its social-acquaintances to locate them, else this field could be NULL.

The role of the matching service is then only to check if there are any PIs of interest to a user that are located in a location grid with the same PL as the mobile user, and if yes, create a trigger to the end-user app notifying of nearby PIs of interest (Step 6). Such a matching can be trivially done, since the matching service knows the set of PLs in which a particular PI is located. Note that because of initial registration phase of the mobile user with the LBS in Step 3, the mobile user can decide which businesses / acquaintances correspond to the notified PIs. Note the matching service knows of (and interacts with) only the PI and PL values of the various entities - never their actual names or locations.

4.2The need for a Matching Service

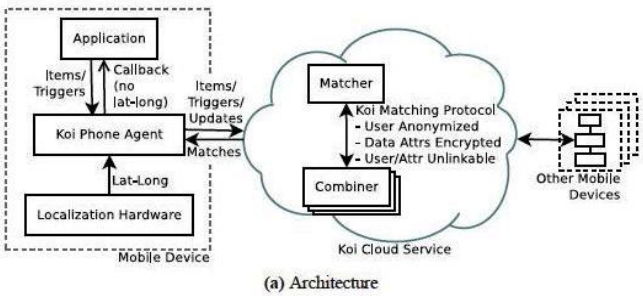
The goal of our paper was to design a system in which no one entity has access to all pieces of sensitive information related to the usage of location based services. Is it possible for entities in a LBS system (mobile user, LBS provider, mobile operator) to encode and exchange information they possess, in a manner that removes the need for a separate matching service? We consider such alternatives, and build a case for why a matching service is essential. LBS exchanges PIs directly with the mobile operator: In this approach, the LBS encodes businesses with PI encodings and notifies the mobile operator of the actual locations corresponding to each PI. The mobile user (as in Step 3 of Figure 4.3) registers with the LBS for interested services and gets the corresponding PIs of its interests. The user's query with the <desired-PI-list> is sent directly to the mobile operator. The mobile operator, based on the current location of the user, returns the PIs that are located in the vicinity of the user. In this case, the mobile operator, based on the PI/location mappings, and the user locations (and their PIs of interest) carries out the match between an user and their interests. However, the mobile operator can decode the PI mapping and infer the user's objects of interest. Suppose the operator has access to a yellow page service though which it can find businesses at any particular location. Consider a business with PI = 385, present at locations L = {L1,L2,L3, ...Ln}. The operators can find the set of businesses B(Li) corresponding to any location Li. Then, if Ti=n i=1 B(Li) corresponds to a single entity (i.e. cardinality of set is one), it can decode the business/interest corresponding to PI = 385. Mobile operator exchanges PLs directly with the LBS: In this approach, there are no business / acquaintance encodings (i.e. no PIs). The operator however assigns location encodings (i.e. PLs) to each location-grid. In order to match businesses, the LBS provider initially gives the mobile operator a list of businesses and their locations, and the operator returns to the LBS, a set (permuted to ensure it does not trivially reveal the PI to location mapping) of PLs associated with each business. In order to locate nearby businesses, a user sends it's PL (that it acquired from the operator) to the LBS, and is notified of businesses whose PL-set contains the user's current PL. In this approach however, the LBS provider can infer the location of an user, by correlating the PL sets corresponding to different businesses, and since it has knowledge of the physical locations of businesses. As shown in Figure 4.1, suppose B1 and B2 share a common location (here (x=2, y=2)). Since the only common entry in PL sets {9, 2, 5} corresponding to B1 and {4, 9} for B2 is 9, the LBS can decipher that the location having PL of 9 is (x=2, y=2).

5. A Location-Privacy Platform for Smartphone Apps

Proposed a privacy-preserving location based matching as a fundamental platform primitive and as an alternative to exposing low-level, latitude-longitude coordinates to applications. Applications set rich location-based triggers and have these be fired based on location updates either from the local device or from a remote device. But issue pertains to malicious applications registering a large number of triggers at sensitive locations, and reverse-engineering a victim user's location based on triggers matched. A weak defense against this attack would be rate-limit to the number of trigger registrations from an application.

5.1 System Architecture

Koi comprises two components, one which runs on the User's mobile device and the other in the cloud (Figure a). The mobile component of Koi interfaces between applications and the cloud component. To applications, it Exposes a simple API (Figure b), which allows registration and updating of *items* and *triggers*, and provides Notification through a callbacks. An *item* is a statement.



(a) Architecture

Registering Items and Triggers
ITEM = CREATEITEM(CONTENT, TTL)
TRIG = CREATETRIGGER(CALLBACK, TTL)
Create a new item or trigger.
Adding Attributes
ADDLOCATTR(ITEMORTRIG, LOC, AUTOUPDATE)
ADDAPPATTR(ITEMORTRIG, ATTR)
Associate location or other attributes to items and triggers.
Notifications
CALLBACK.NOTIFY(CONTENT)
Called by platform when a match is found.

(b) API

5.2 Platform API

The platform service model is similar to a database trigger. The app registers items with the Koi phone-agent. Items may correspond to users or content (e.g., photos). The app associates attributes with an item. Attributes may be locations, keywords, or arbitrary data. The app also registers triggers. Triggers specify one or more attributes that must match. When an item matching the trigger is registered (by another user or app, or by the same app), the app registering the trigger is notified of the item through a callback. Figure b lists the Koi API. The app specifies location attributes symbolically (e.g., loc: self, or within 1 block of loc: self). The Koi phone-agent internally replaces loc: self with the actual at-long. The agent optionally automatically updates the lat-long if the user's location changes. This amortizes the Cost of acquiring user location across multiple apps, and avoids having to wake each app up whenever the user moves. By never exposing lat-long data to the app, Koi minimizes the app accidentally leaking it to third-parties. The app may associate arbitrary content with an item. A social networking app may, for instance, register the user's push-notification service handle so another user can contact this user. A citizen-journalism app may, for instance, register a photo or URL etc. The content may additionally be encrypted, for instance in the social networking app, only friends with the appropriate key may recover the push-notification handle.

4.2 Privacy-Preserving Matching Service

The operation of the Koi cloud service is best illustrated through an example. Consider the scenario in Table 1 where users Alice and Chuck register an item indicating they are tour-guides for Bangalore and Boston respectively. Bob registers a trigger looking for a tour-guide at his present location. The goal is to match Bob, who happens to be in Bangalore, with Alice. Note that Bob's phone-agent uses his lat-long without first geocoding it to Bangalore. For simplicity, we assume each of them register some unique user ID (as the item content, or the trigger callback) through which they can be reached. Our location-privacy goal is to prevent the cloud service from associating this user ID with the user's location.

6. Analysis of all the Review Techniques

Privacy preserving route planning [2] proposed Secure Multiparty Computation (SMC) protocols for securely computing the distance between the points. One difficulty with route planning protocols is the requirement that the device know where it is at, which would seem to require some form of query to a GPS system, but this would reveal the location of the device.

Where shall we meet? Proposing optimal locations for meetings [3] proposing optimal locations for meetings presented a survey of existing literature on meeting-location algorithms and proposes a more comprehensive solution for such a problem. The system, while useful, may be complicated for some users. Automating system defaults when users provide insufficient data from calendars or start points can help, but preferences about times, venues, and travel methods can be complicated even when known. An organizer, or participants who vote, need to evaluate choices and fine-tune results to suit group criteria.

Trust no one [4] a decentralized matching service for privacy in location based services," suggested a novel approach to deploy location-based services in which user privacy is guaranteed. Inspite of operating on encoded information got from the operator and the LBS; it is able to trigger updates to the mobile user whenever the user is in the same location of its interested services.

Koi: A location-privacy platform for smartphone apps,[5] proposed a privacy-preserving location based matching. But issue pertains to malicious applications registering a large number of triggers at sensitive locations, and reverse-engineering a victim user's location based on triggers matched. A weak defense against this attack would be rate-limit to the number of trigger registrations from an application.

We plan to improve the execution time of PPFRVP and to investigate more efficient designs that reduce communication between devices and interactions between users, without sacrificing security or usability. We also plan to investigate ways to recover from errors and attacks such that after detecting an error in a subgroup, the entire group does not need to rerun the protocol.

References

[1] Igor Bilogrevic, Murtuza Jadliwala, Vishal Joneja, Kübra Kalkan, Jean-Pierre Hubaux, and Imad Aad, "Privacy-Preserving Optimal Meeting Location Determination on Mobile Devices" IEEE Transactions on Information Forensics and Security, vol. 9, no. 7, pp. 1141-1156, JULY 2014.
[2] K. B. Frikken and M. J. Atallah "Privacy preserving route planning," in Proc. ACM WPES, pp. 8– 15, 2004.
[3] P.Santos and H.Vaughn, "Where shall we meet? Proposing optimal locations for meetings," in Proc. MapISNet, 2007.
[4] S. Jaiswal and A. Nandi, "Trust no one: A decentralized matching service for privacy in location based services," Proc. ACM MobiHeld, 2010.
[5] S. Guha, M. Jain, and V. Padmanabhan, "Koi: A location privacy platform for smartphone apps," Proc. 9th USENIX Conf. NSDI, 2012.